Spud, flml and Diamond:

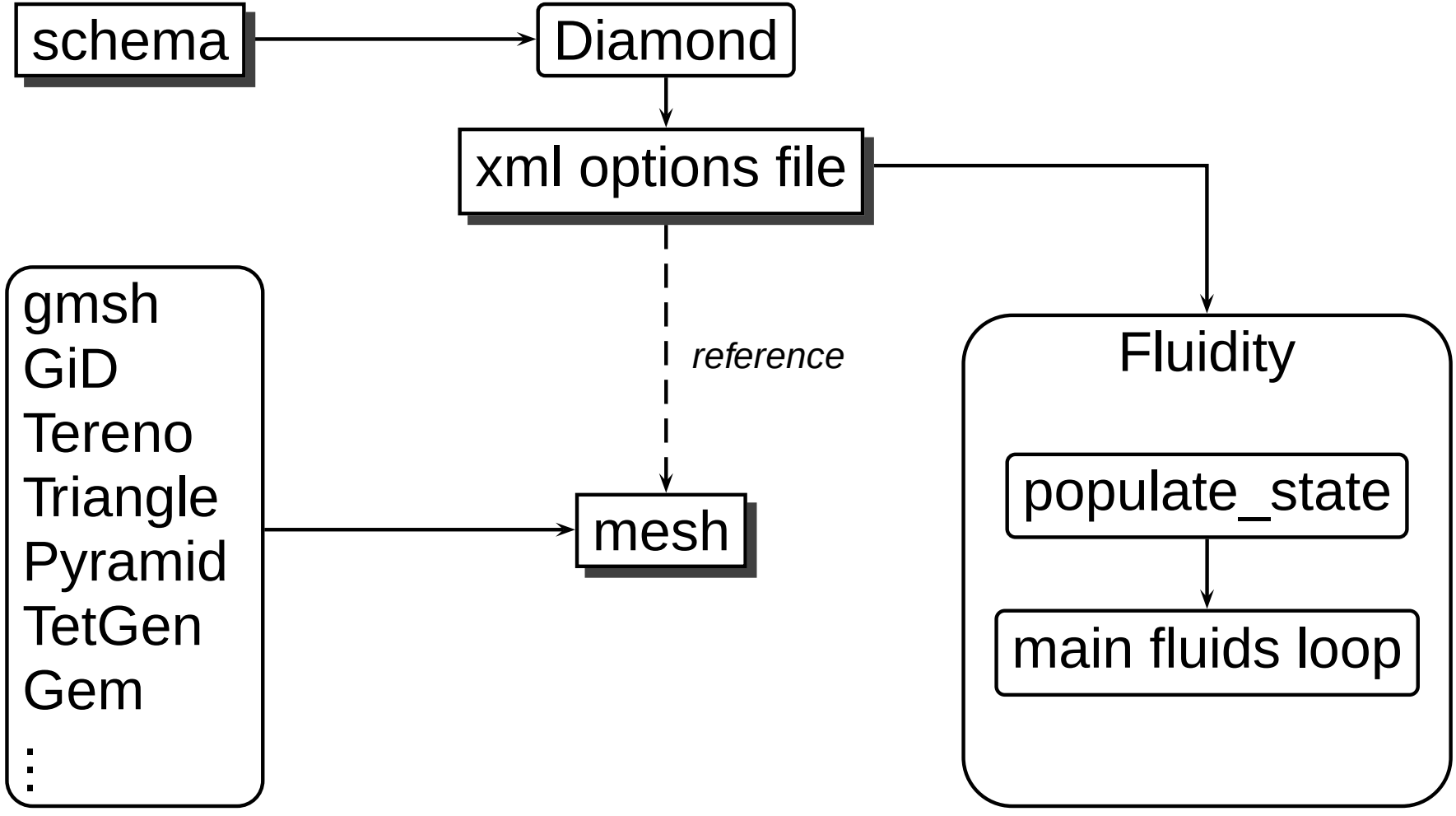
the new Fluidity front end.

David Ham, Patrick Farrell, Gerard Gorman, Cian Wilson, James Maddison, Stephan Kramer, Jemma Shipton, Colin Cotter, Gareth Colins, Matthew Piggott and Jefferson Gomes

David.Ham@imperial.ac.uk

Imperial College London

The new fluidity frontend



The options problem

1. Defining options
2. Setting options
3. Accessing options from the code

Defining the options file

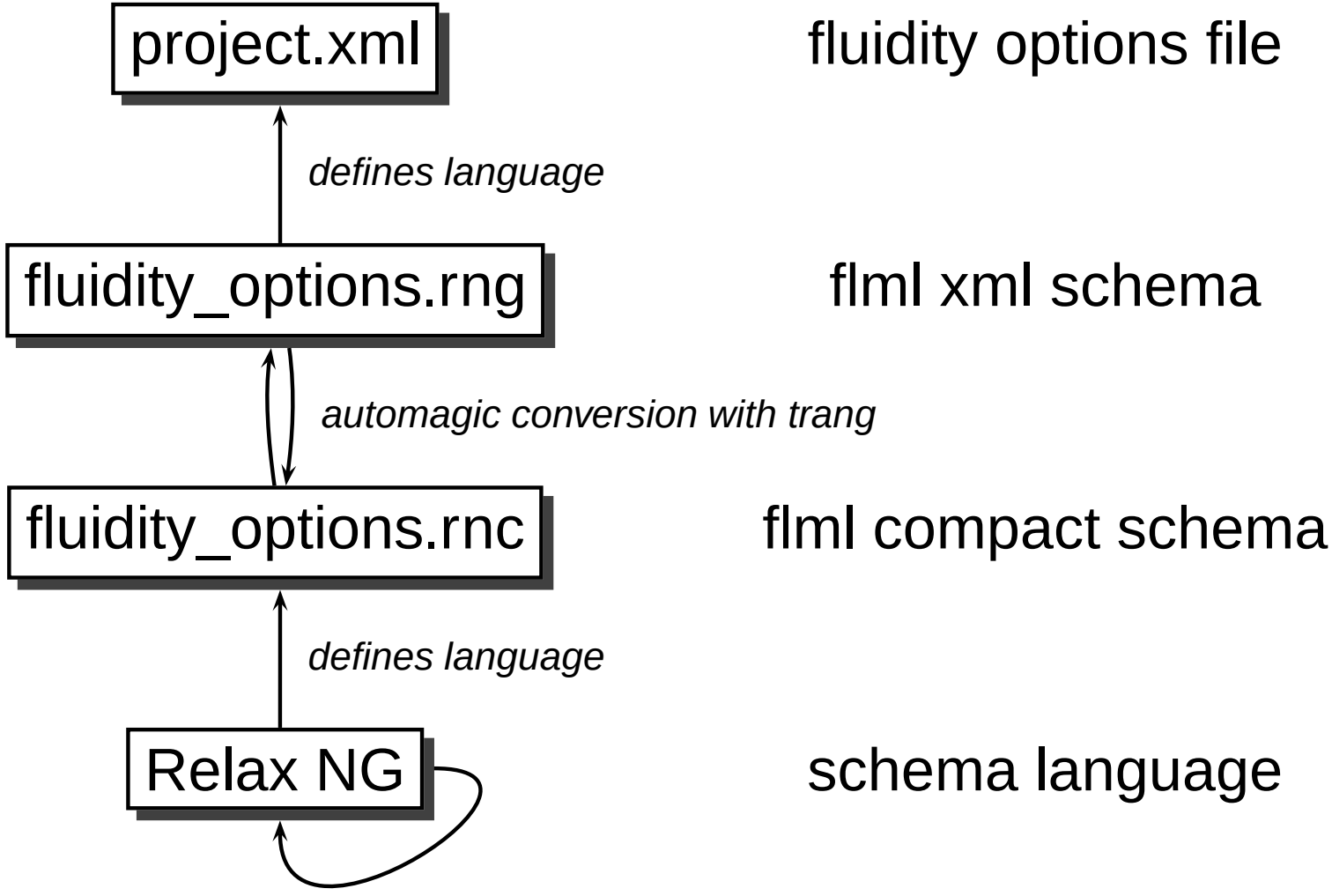
The options file is written in an xml language called flml (Fluidity Markup Language). xml (extensible markup language) is:

1. plain text
2. machine parseable
3. internationally standard
4. markup!

A brief xml example

```
<timestepping>
  <current_time replaces="ACCTIM">
    <real_value rank="0">0</real_value>
  </current_time>
  <finish_time replaces="LTIME">
    <real_value rank="0">6.283185307179586</real_value>
    <comment>2 * pi (ie one rotation of the fluid)</comment>
  </finish_time>
  <timestep replaces="DT">
    <real_value rank="0">0.12566370614359174</real_value>
    <comment>2 * pi/50 (50 steps per revolution)</comment>
  </timestep>
</timestepping>
```

Definition chain



A compact schema fragment.

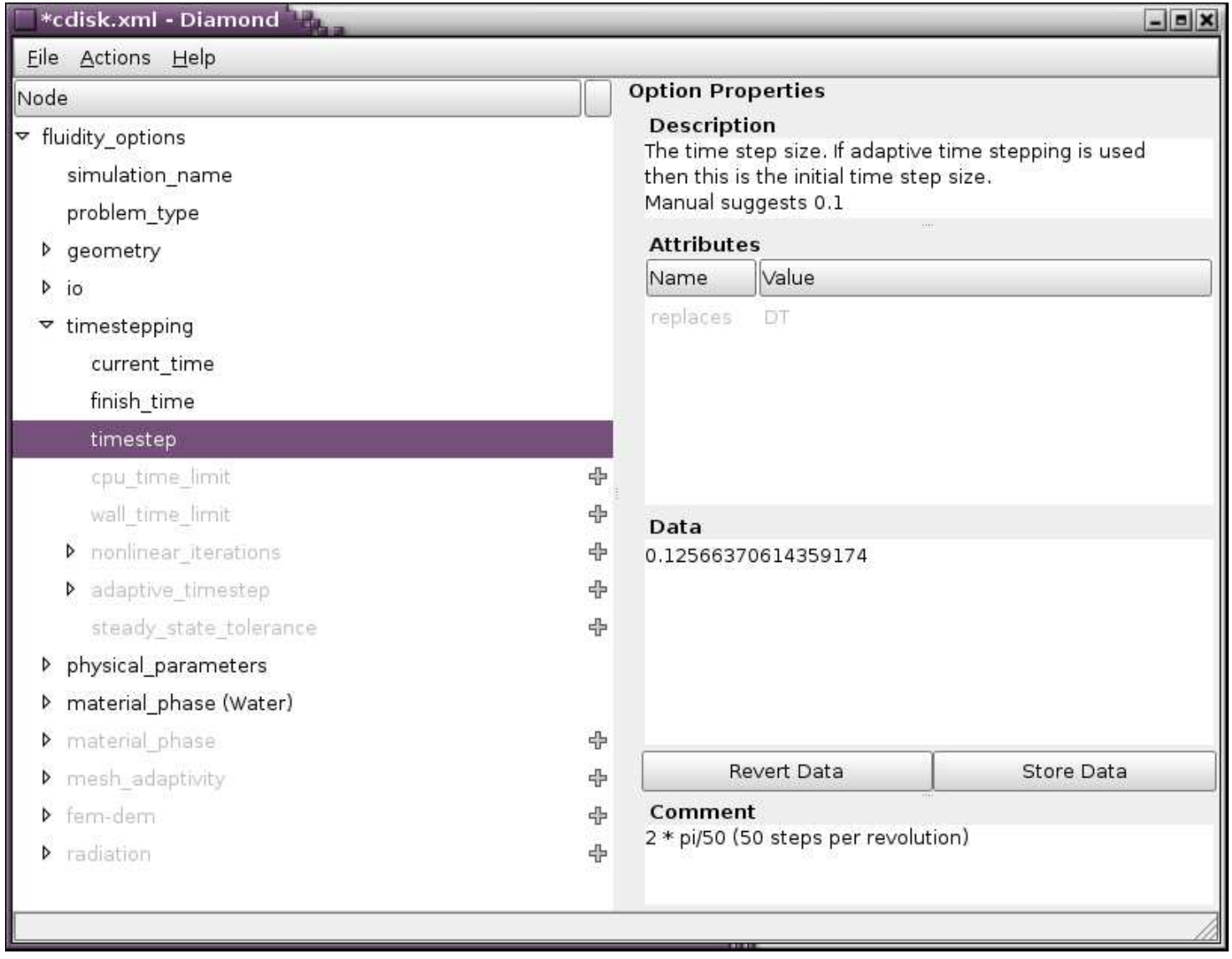
```
## Options dealing with time discretisation
element timestepping {
   ## Current simulation time. At the start of the simulation this
   ## is the start time.
   element current_time {
      attribute replaces {"ACCTIM"},
      real
   },
   ## Simulation time at which the simulation should end.
   element finish_time {
      attribute replaces {"LTIME"},
      real
   },
   ## The time step size. If adaptive time stepping is used
   ## then this is the initial time step size.
   ## Manual suggests 0.1
   element timestep {
      attribute replaces {"DT"},
      real
   },
```

The corresponding xml schema

```
<element name="timestepping">
  <a:documentation>Options dealing with time discretisation</a:documenta
  <element name="current_time">
    <a:documentation>Current simulation time. At the start of the simula
is the start time.</a:documentation>
    <attribute name="replaces">
      <value>ACCTIM</value>
    </attribute>
    <ref name="real"/>
  </element>
  <element name="finish_time">
    <a:documentation>Simulation time at which the simulation should end.<
    <attribute name="replaces">
      <value>LTIME</value>
    </attribute>
    <ref name="real"/>
  </element>
```

... and the timestep doesn't even fit on the page.

Diamond: the automatic GUI



XML output (reminder)

```
<timestepping>
  <current_time replaces="ACCTIM">
    <real_value rank="0">0</real_value>
  </current_time>
  <finish_time replaces="LTIME">
    <real_value rank="0">6.283185307179586 2</real_value>
    <comment>2 * pi (ie one rotation of the fluid)</comment>
  </finish_time>
  <timestep replaces="DT">
    <real_value rank="0">0.12566370614359174</real_value>
    <comment>2 * pi/50 (50 steps per revolution)</comment>
  </timestep>
</timestepping>
```

Accessing options in fluidity

Fluidity (recent versions: svn up is your friend!) can be told to read an xml input file with the --xml option:

```
> dfluidity --xml cdisk.xml
```

This causes the xml to be read into a *dictionary* which can be accessed from anywhere in the code:

```
call get_option("/timestepping/timestep", dt)
```

The dictionary interface

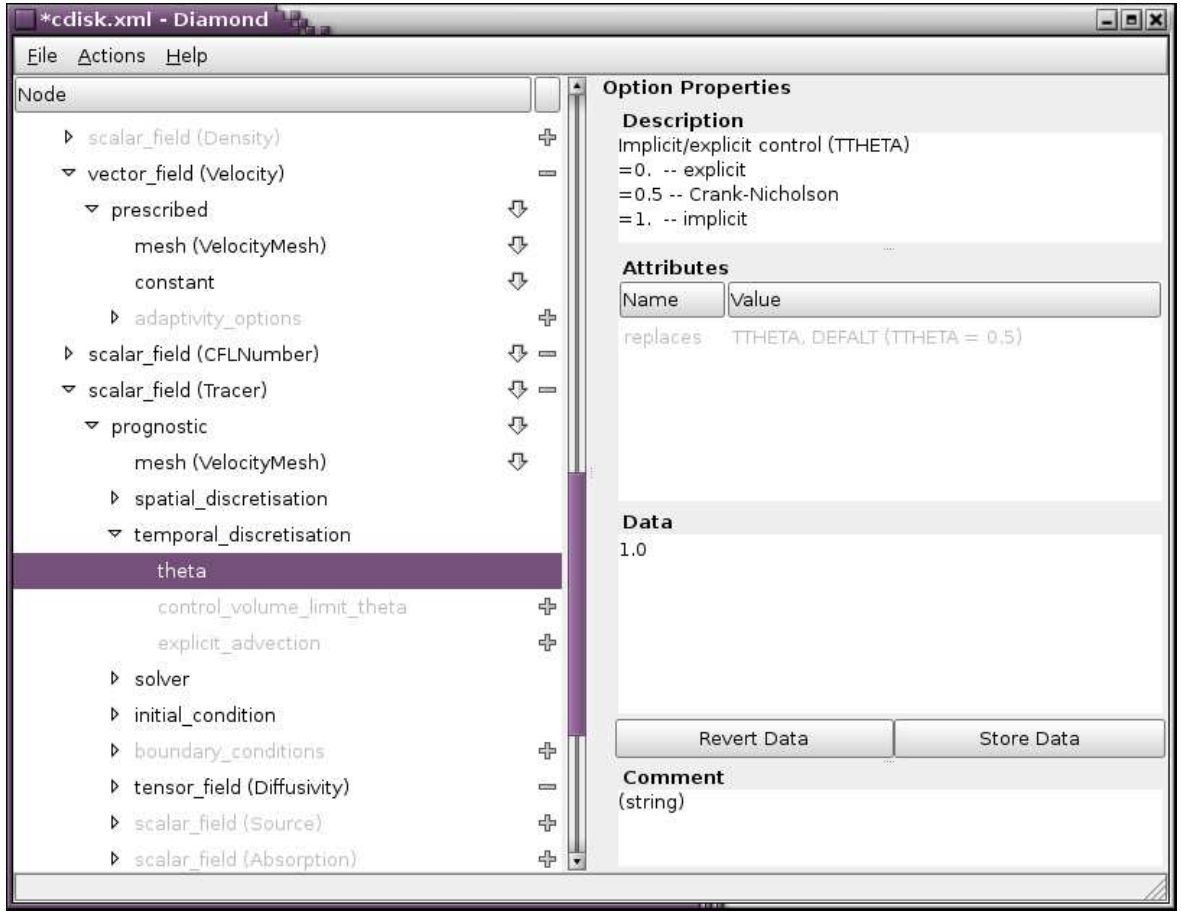
```
module options

subroutine get_option(key, option, stat, default)
    !!< Return the value of the option given by key
    character(len=*), intent(in) :: key
    real, dimension(:), intent(out) :: option
    <TYPE>, intent(out), optional :: stat
    real, intent(in), optional :: default

function have_option(key)
    !!< Test for the presence of the option given by key.
    logical :: have_option
    character(len=*), intent(in) :: key
```

Options and fields

Many options are associated with the solution of particular fields. These options are repeated for each (appropriate) field in the tree:



There are other repeated options repeated in a similar manner.

Accessing Fields

New field data types carry their options path with them:

```
call get_option(trim(T%option_path)//&
     &"/prognostic/temporal_discretisation/theta", theta)
```

Back in old fluidity land, you can look up the path from the field index in T:

```
if(have_option(trim(field_optionpath_list(it))//
   & "/prognostic/spatial_discretisation/discontinuous_galerkin")) then
   ! Solve the DG form of the equations.
```

Populate_state

A further task which used to be performed by Gem was the population of initial and boundary values. This is now performed by populate_state.

Species of field

prognostic fields Anything we solve a PDE for. Eg Velocity, Temperature, Pressure...

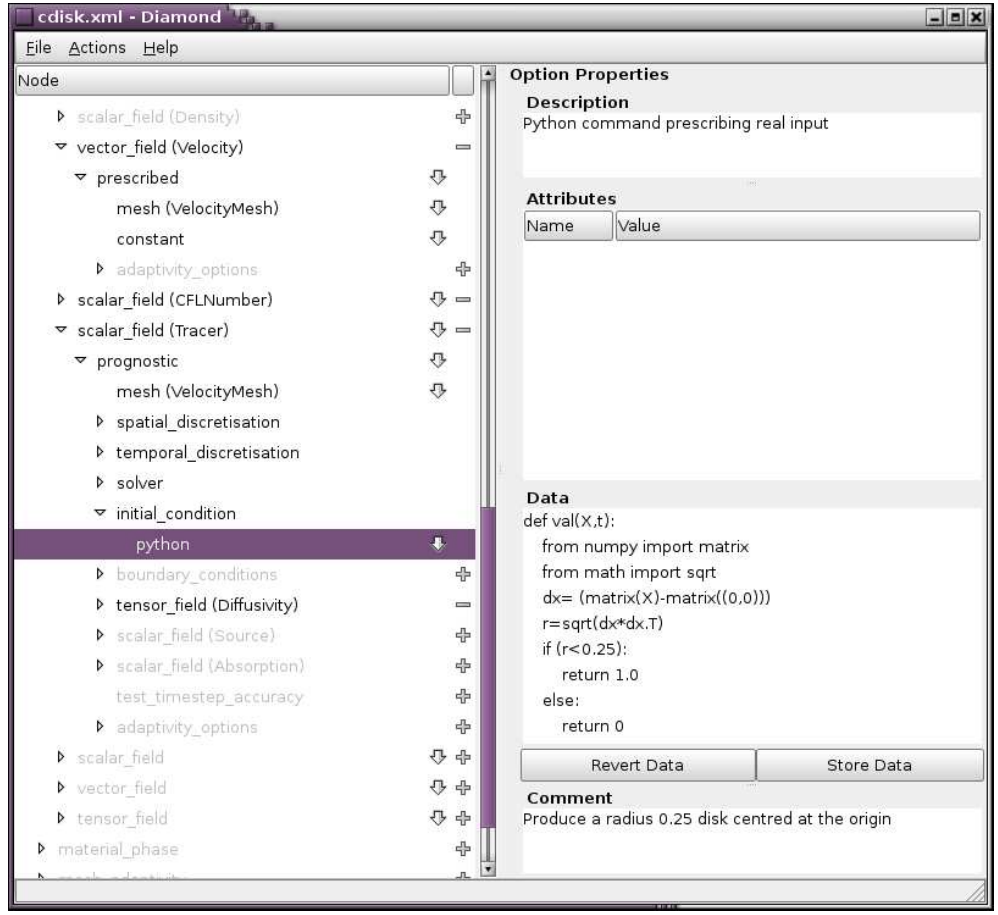
diagnostic fields Calculated during execution by the model based on the system state. Eg. CFL Number

prescribed fields Fields whose value comes from outside the model eg. climatology data or a prescribed function.

aliased fields Like a symlink to another field. Used, for example, when the same field appears in more than one materal or phase.

Prescribing fields using python

The initial value of a prognostic field and the value of a prescribed field can be set in the GUI by specifying a Python function:



Python function syntax

The Python code provided to set a field must define a function val whose arguments are a position vector X and the time t. The function returns the field value at that point. For example:

```
def val(X,t):
  from numpy import matrix
  from math import sqrt
  dx= (matrix(X)-matrix((0,0)))
  r=sqrt(dx*dx.T)
  if (r<0.25):
    return 1.0
  else:
    return 0
```

Python for setting vector fields

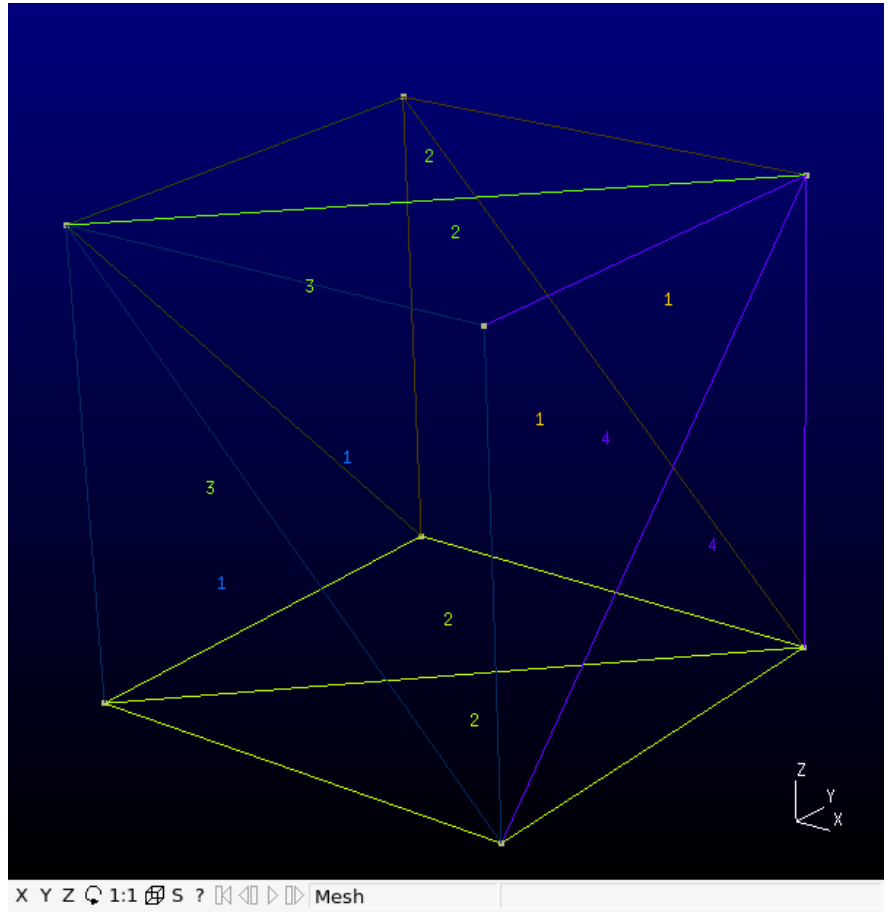
The python function for setting a vector field naturally has to return a vector quantity as in this routine for setting a solenoidal velocity field:

```
def val(X,t):
  # Rigid rotation about origin.
  return (X[1], -X[0])
```

Setting tensor valued fields from Python will be supported when someone provides enough liquid bribes!

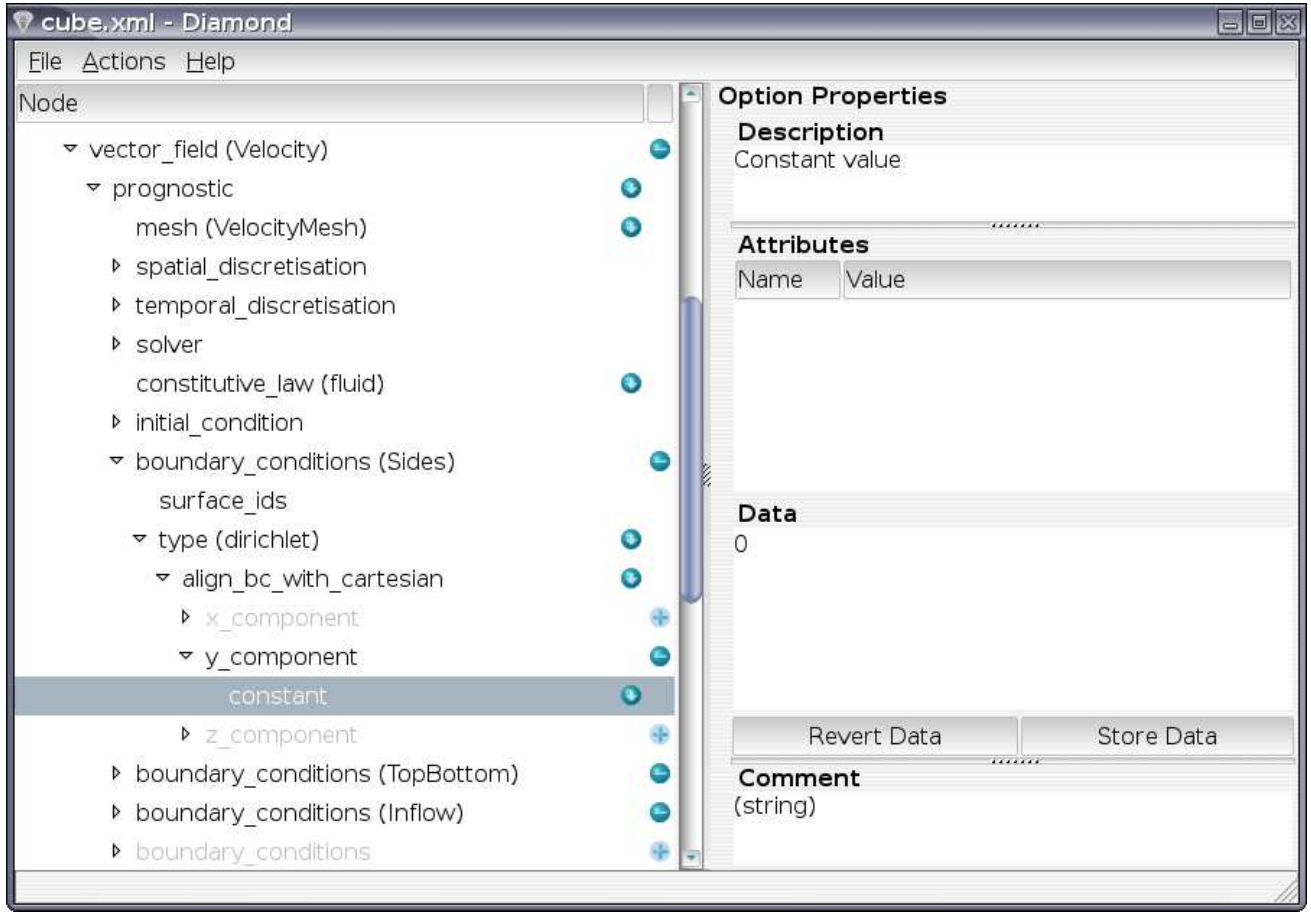
Boundary regions

Mesh boundary regions are labelled. You can have any number of these (in most mesh generators). Each boundary condition for each mesh applies to a list of these.



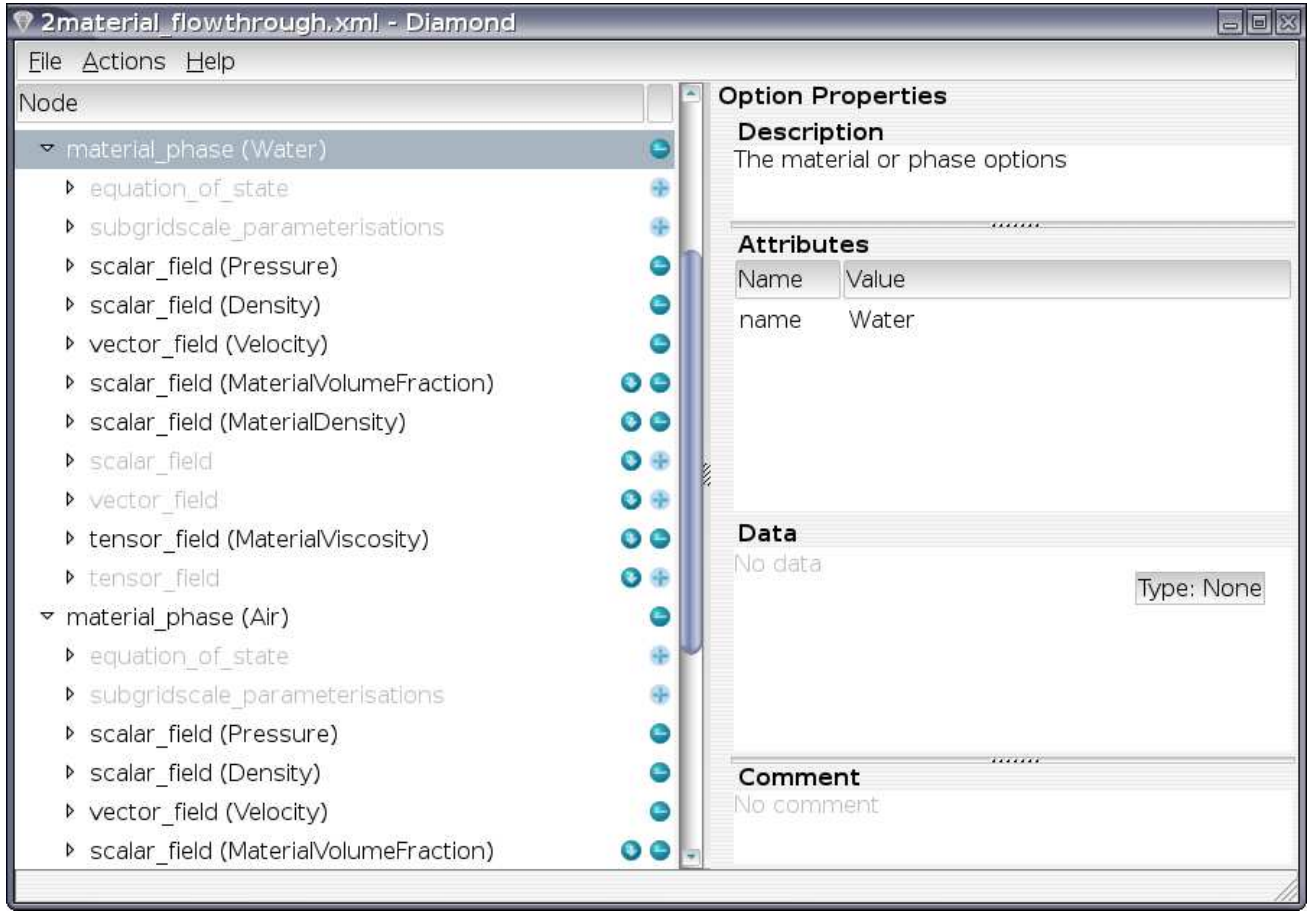
Setting boundary conditions

Each component of each boundary condition can be set as a constant, a python function or as a generic function:



Multiple material phases

Each material or phase is one material_phase. A material_phase groups fields associated with one material or phase:



State of play

1. Vast majority of used options are in the schema.
2. Diamond is basically complete.
3. Some problems are known to run.
4. Boundary conditions are partially done (most existing functionality is there).
5. Multimaterial works.

To do...

1. Parallel
2. Hexes
3. Multiphase
4. Radiation
5. Adjoint